## Uppy development

Fork the repository into your own account first. See the [GitHub Help](https://help.github.com/articles/fork-a-repo/) article for instructions.

After you have successfully forked the repo, clone and install the project:

```bash

git clone git@github.com:YOUR\_USERNAME/uppy.git

cd uppy

npm install

```

Our website’s examples section is also our playground, please read the [Local Previews](#Local-previews) section to get up and running.

### Requiring files

- If we are `require()`ing a file from the same subpackage (e.g. require `@uppy/dashboard/utils/hi.js` from `@uppy/dashboard/src/index.js`) - we can freely use relative imports, as long as the required file is under the `src` directory (`/:packageName/src/\*\*/\*.js`).

- But if we want to require some file from another subpackage - we should use global @uppy requires, and they should always be in the form of `@uppy/:packageName/(lib instead of src)/(same path).js`

## Tests

Unit tests are using Jest and can be run with:

```bash

npm run test:unit

```

For end-to-end tests, we use [Webdriverio](http://webdriver.io). For it to run locally, you need to install a Selenium standalone server. Just follow [the guide](http://webdriver.io/guide.html) to do so. You can also install a Selenium standalone server from NPM:

```bash

npm install selenium-standalone -g

selenium-standalone install

```

And then launch it:

```bash

selenium-standalone start

```

After you have installed and launched the selenium standalone server, run:

```bash

npm run test:endtoend:local

```

By default, `test:endtoend:local` uses Firefox. You can use a different browser, like Chrome, by passing the `-b` flag:

```bash

npm run test:endtoend:local -- -b chrome

```

> Note: The `--` is important, it tells npm that the remaining arguments should be interpreted by the script itself, not by npm.

You can run in multiple browsers by passing multiple `-b` flags:

```bash

npm run test:endtoend:local -- -b chrome -b firefox

```

When trying to get a specific integration test to pass, it's not that helpful to continuously run \_all\_ tests. You can use the `--suite` flag to run tests from a single `./test/endtoend` folder. For example, `--suite thumbnails` will only run the tests from `./test/endtoend/thumbnails`. Of course, it can also be combined with one or more `-b` flags.

```bash

npm run test:endtoend:local -- -b chrome --suite thumbnails

```

These tests are also run automatically on Travis builds with [SauceLabs](https://saucelabs.com/) cloud service using different OSes.

## Releases

Before doing a release, check that the examples on the website work:

```bash

npm start

open http://localhost:4000/examples/dashboard

```

Also check the other examples:

```bash

cd examples/EXAMPLENAME

npm install

npm start

```

Releases are managed by [Lerna](https://github.com/lerna/lerna). We do some cleanup and compile work around releases too. Use the npm release script:

```bash

npm run release

```

If you have two-factor authentication enabled on your account, Lerna will ask for a one-time password. There is an issue with the CLI where the OTP prompt may be obscured by a publishing progress bar. If Lerna appears to hang just as it starts publishing, chances are it's waiting for the password. Try typing in your OTP and hitting enter.

Other things to keep in mind during release:

\* When doing a major release >= 1.0, of the `@uppy/core` package, the `peerDependency` of the plugin packages needs to be updated first. Eg when updating from 1.y.z to 2.0.0, the peerDependency of each should be `"@uppy/core": "^2.0.0"` before doing `npm run release`.

\* When adding a new package, add the following key to its package.json:

```json

"publishConfig": { "access": "public" }

```

Else, npm will try and fail to publish a \_private\_ package, because the `@uppy` scope on npm does not support that.

After a release, the demos on transloadit.com should also be updated. After updating, check that some things work locally:

- the demos in the demo section work (try one that uses an import robot, and one that you need to upload to)

- the demos on the homepage work and can import from GDrive, Insta, Dropbox

If you don't have access to the transloadit.com source code ping @arturi or @goto-bus-stop and we'll pick it up. :sparkles:

## Website development

We keep the [uppy.io](http://uppy.io) website in `./website`, so it’s easy to keep docs and code in sync as we are still iterating at high velocity.

The site is built with [Hexo](http://hexo.io/), and Travis automatically deploys this onto GitHub Pages (it overwrites the `gh-pages` branch with Hexo's build at every change to `master`). The content is written in Markdown and located in `./website/src`. Feel free to fork & hack!

Even though bundled in this repo, the website is regarded as a separate project. As such, it has its own `package.json` and we aim to keep the surface where the two projects interface as small as possible. `./website/update.js` is called during website builds to inject the Uppy knowledge into the site.

### Local previews

1. `npm install`

1. `npm start`

1. Go to http://localhost:4000. Your changes in `/website` and `/packages/@uppy` will be watched, your browser will refresh as files change.

Then, to work on, for instance, the XHRUpload example, you would edit the following files:

```bash

${EDITOR} packages/@uppy/core/src/index.js \

packages/@uppy/core/src/Plugin.js \

packages/@uppy/xhr-upload/src/index.js \

website/src/examples/xhrupload/app.es6

```

And open <http://localhost:4000/examples/xhrupload/> in your web browser.

## CSS guidelines

The CSS standards followed in this project closely resemble those from [Medium's CSS Guidelines](https://gist.github.com/fat/a47b882eb5f84293c4ed). If something is not mentioned here, follow their guidelines.

### Naming conventions

This project uses naming conventions adopted from the SUIT CSS framework.

[Read about them here](https://github.com/suitcss/suit/blob/master/doc/naming-conventions.md).

To quickly summarize:

#### Utilities

Syntax: `u-[sm-|md-|lg-]<utilityName>`

```css

.u-utilityName

.u-floatLeft

.u-lg-col6

```

#### Components

Syntax: `[<namespace>-]<ComponentName>[-descendentName][--modifierName]`

```css

.twt-Button /\* Namespaced component \*/

.MyComponent /\* Components pascal cased \*/

.Button--default /\* Modified button style \*/

.Button--large

.Tweet

.Tweet-header /\* Descendents \*/

.Tweet-bodyText

.Accordion.is-collapsed /\* State of component \*/

.Accordion.is-expanded

```

### SASS

This project uses SASS, with some limitations on nesting. One-level-deep nesting is allowed, but nesting may not extend a selector by using the `&` operator. For example:

```sass

/\* BAD \*/

.Button {

&--disabled {

...

}

}

/\* GOOD \*/

.Button {

...

}

.Button--disabled {

...

}

```

### Mobile-first responsive approach

Style to the mobile breakpoint with your selectors, then use `min-width` media queries to add any styles to the tablet or desktop breakpoints.

### Selector, rule ordering

- All selectors are sorted alphabetically and by type.

- HTML elements go above classes and IDs in a file.

- Rules are sorted alphabetically.

```sass

/\* BAD \*/

.wrapper {

width: 940px;

margin: auto;

}

h1 {

color: red;

}

.article {

width: 100%;

padding: 32px;

}

/\* GOOD \*/

h1 {

color: red;

}

.article {

padding: 32px;

width: 100%;

}

.wrapper {

margin: auto;

width: 940px;

}

```